Amendments to the Specification:

Please replace paragraph [0015], beginning at page 3, line 19, with the following amended paragraph:

[0015] The rename stage 16 may assign a new physical register to destination operands, and map source operands of subsequent instructions onto the corresponding physical registers. The rename stage 16 determines whether the source operands needed by an instruction currently reside in the physical register file, or whether the instruction needs to wait for another instruction in the scheduling window to produce the operand. The rename stage 16 outputs the physical register number of the source operand. As the rename stage 16 determines ~~dependences~~ dependencies in terms of physical register numbers, it may also calculate the ~~dependences~~ dependencies in terms of scheduler entry numbers.

Please replace paragraph [0017], beginning at page 4, line 13, with the following amended paragraph:

[0017] After instructions are renamed, they may be placed in a baseline scheduler 30. The scheduler 30 is responsible for

issuing instructions to the execution units when the required resources (source operands and execution units) are available. The scheduler 30 may be primarily comprised of the wakeup arrays 32a and 32b and the select logic 34a and 34b. The wakeup arrays 32a and 32b may hold executed instructions as well as instructions that have not executed. [[,]] The select logic 34a and 34b picks instructions for execution. Typically, each wakeup array 32a and 32b feeds a separate select logic circuit 34a and 34b.

Please replace paragraph [0023], beginning at page 7, line 11, with the following amended paragraph:

[0023] Figures 3-5 show the baseline scheduling logic in greater detail. The scheduling logic may be comprised of wakeup arrays 32, selectors 34, and countdown timers (not shown). As shown in Figure 4, each ~~Each~~ wakeup array entry 50 preferably contains the wakeup logic for a single instruction. The wakeup logic may be implemented with wire-OR-style wakeup logic as well as CAM-style wakeup logic. Each entry 50 may contain a bit vector, called a Resource Vector 52, that indicates which resources the instruction needs. Each bit position, or Resource Bit 54, within the Resource Vector 52 corresponds to a

particular resource. A resource can be either a result operand produced by the instruction in a particular entity of a wakeup array 32, or a particular functional unit. Each Resource Bit 54 is set if the instruction requires that resource, and reset if the instruction does not.

Please replace paragraph [0024], beginning at page 8, line 1, with the following amended paragraph:

[0024] Figures 3 and 4 show a dependency graph 58 and an example of a wakeup array 32 that contains the instructions in the graph. The portion of the wakeup array 32 that is shown has four Resource Vectors 52 with seven Resource Bits 54. The instructions in the wakeup array entries are the SHIFT, SUB, ADD, and MULT instructions from the dependency graph. In this example, the instructions that produced the values for the unspecified source operands of the SHIFT, [[.]] SUB, ADD, and MULT instructions have already executed, so their result values reside in the register file 44. The SHIFT instruction only requires the shifter, so only one Resource Bit 54 is set. The SUB and ADD instructions depend on the result of the SHIFT and require the ALU. The MULT instruction depends on the result of the SUB and requires the multiplier.

Please replace paragraph [0025], beginning at page 8, line 15, with the following amended paragraph:

[0025] Figure 5 shows the wakeup logic 60 for one wakeup array entry 50. The AVAILABLE lines 64 running vertically pass through every entity in the array 32. Each line corresponds to a Resource Bit 54 in the Resource Vector 52. The line is high if the resource is available and low if it is not. The SCHEDULED bit 62 indicates whether or not the instruction has been granted execution. There may be a number of cycles between the time the instruction is granted execution and the time the instruction is actually de-allocated from the wakeup array 32. During this time, the SCHEDULED bit 62 is set to prevent the instruction from requesting execution again. If the instruction is rescheduled, for example, due to a load latency misprediction, the SCHEDULED bit 62 is reset by asserting the Reschedule line. The instruction requests execution if 1) the SCHEDULED bit is not set, and 2) for each resource, the instruction does not require that resource or that resource is available. The AND gate may be implemented using a wire-OR structure to make it fast. Hence this style of wakeup logic is called wire-OR-style.

Please replace paragraph [0026], beginning at page 9, line 8, with the following amended paragraph:

[0026] The select logic 34 may be a priority circuit. The input is a bit vector indicating which instructions from the wakeup array 32 ~~re-quest~~ request execution. One of the outputs of the select logic 34 is the Grant Vector 38, indicating which instructions receive the execution grants. The wakeup array 32 uses the Grant Vector 38 to set the SCHEDULED bits 62. The other outputs are a set of one-hot bit vectors. The first one-hot specifies the first instruction that received an execution grant. The second one-hot specifies the second instruction that received an execution grant. And so on. For a select-l priority circuit, there is only 1 one-hot vector, and it may be equivalent to the Grant Vector 38. Each one-hot is used to access a port of a Payload RAM 42 and deliver the payload for the associated instruction to the register file 44 and to a functional unit, FU, 47. The one-hot may be the set of word lines for the Payload RAM 42, so that the Payload RAM 42 does not require a word decoder.

Please replace paragraph [0027], beginning at page 10, line 3, with the following amended paragraph:

[0027] After an instruction receives an execution grant, the AVAILABLE lines for the associated wakeup array entries 52 are asserted so that the dependent instructions may wake up. For a single-cycle instruction, the AVAILABLE line may be asserted immediately. For an N-cycle instruction, the AVAILABLE line may be asserted [[N- 1]] N-1 cycles later. This may be accomplished by using a countdown timer initialized to the latency of the instruction. When an instruction receives an execution grant, the timer begins to count down. When the timer completes the countdown, the AVAILABLE line of the instruction may be asserted.

Please replace paragraph [0032], beginning at page 12, line 11, with the following amended paragraph:

[0032] A collision is the scenario where more instructions wakeup than can be selected, resulting from an incorrect speculation by at least one instruction: any[[. Any]] unselected instructions assert their AVAILABLE lines too early. These unselected instructions are called collision victims. Collision

victims may be identified at the same time an instruction is selected. For example, when a Grant Vector is produced, a collision victim vector 112 may also be produced. Dependents of the collision victims may also wake up before they are really ready to be scheduled, thus entering the scheduling pipeline too early. We call these instructions pileup victims. Pileup victims may be identified by a scoreboard check before the execute stage.

Please replace paragraph [0040], beginning at page 15, line 17, with the following amended paragraph:

[0040] Select-N schedulers can select more than one instruction per cycle. For select-1 schedulers, there is a collision when 2 or more instructions request execution. For select-2 schedulers, there is a collision when 3 or more instructions request execution. As the number of instructions selected increases and the total number of schedulers decreases, the probability of a collision decreases. To demonstrate this, three machines were simulated, each with eight functional units and the same size scheduling window. The first had eight select-1 schedulers, the second had four select-2 schedulers, and the third had two select-4 schedulers. For an average cycle, the

probability of a collision in any scheduler for the machine with ~~select I~~ select-1 schedulers was 39%, for the machine with select-2 schedulers was 26%, and for the machine with select-4 schedulers was 15%. Although select-2 and select-4 logic are more complex than select-1 logic, select-free scheduling allows this logic to be pipelined with little loss in IPC.

Please replace the abstract with the following amended abstract:

A processor having select-free scheduling separates the wakeup and select logic into two loops. A wakeup loop holds scheduler instructions including unexecuted instructions, and indicates which of the unexecuted instructions ~~that~~ may be ready to be executed. At least one of the unexecuted instructions is to wakeup and notify at least another of the unexecuted instructions to speculatively wakeup. A select loop selects at least one of the indicated ready instructions for execution.

~~10128865.doc~~